What’s New in LLDB

Debug your way to fame and glory

Session 402

Kate Stone  Software Behavioralist
Sean Callanan  Master of Expressions
Enrico Granata  Data Wizard
The Year’s Highlights
Changes big and small
Since WWDC 2014
Since WWDC 2014

Shipped Swift debugger
Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
• LLDB in disguise
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
• LLDB in disguise
Numerous improvements
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
• LLDB in disguise
Numerous improvements
• Swift types show inherited Objective-C fields
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
• LLDB in disguise
Numerous improvements
• Swift types
• Help includes command aliases
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
• LLDB in disguise
Numerous improvements
• Swift types
• Help includes command aliases
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
• LLDB in disguise
Numerous improvements
• Swift types
• Help includes command aliases
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases

help
expression -O --
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
  • LLDB in disguise
Numerous improvements
  • Swift types
  • Help includes command aliases
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
  • LLDB in disguise
Numerous improvements
  • Swift types
  • Help includes command aliases
  • Improved data formatting
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL

• LLDB in disguise

Numerous improvements

• Swift types
• Help includes command aliases
• Improved data formatting
  - Set<T>, NSIndexPath

-expression -O -- po

help
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
• LLDB in disguise
Numerous improvements
• Swift types
• Help includes command aliases
• Improved data formatting
  - Set<T>, NSIndexPath
• printf() prototype for expressions
Since WWDC 2014

Shipped Swift debugger
Shipped Swift REPL
  • LLDB in disguise
Numerous improvements
  • Swift types
  • Help includes command aliases
  • Improved data formatting
    - `Set<T>, NSIndexPath`
  • `printf()` prototype for expressions
  • Improved disassembly
Recent Breakpoint Enhancements

Named breakpoints
• Multiple non-unique names allowed
• Other breakpoint commands take names
Recent Breakpoint Enhancements

Named breakpoints

• Multiple non-unique names allowed
• Other breakpoint commands take names
Recent Breakpoint Enhancements

Named breakpoints
- Multiple non-unique names allowed
- Other breakpoint commands take names

- `breakpoint set -N name`
- `breakpoint enable name`
- `breakpoint disable name`
- `breakpoint delete name`
Recent Breakpoint Enhancements

Named breakpoints
• Multiple non-unique names allowed
• Other breakpoint commands take names

Breakpoints in ~/.lldbinit
• Create default breakpoints when LLDB starts
• Inherited by all debug targets

```bash
breakpoint set -N name
breakpoint enable name
breakpoint disable name
breakpoint delete name
```
Recent Breakpoint Enhancements

Named breakpoints
- Multiple non-unique names allowed
- Other breakpoint commands take names

Breakpoints in `~/.lldbinit`
- Create default breakpoints when LLDB starts
- Inherited by all debug targets

```
breakpoint set -n malloc -N memory
breakpoint set -n free -N memory
breakpoint disable memory
```
Xcode 7
Xcode 7

Improvements in expression evaluation

• Swift 2 support
• Objective-C module support
Xcode 7

Improvements in expression evaluation
• Swift 2 support
• Objective-C module support

Improved data formatters
• Vector types in Objective-C and Swift
• Custom data formatters written in Swift
Xcode 7

Improvements in expression evaluation
  • Swift 2 support
  • Objective-C module support
Improved data formatters
  • Vector types in Objective-C and Swift
  • Custom data formatters written in Swift
Address Sanitizer integration
Xcode 7

Improvements in expression evaluation
• Swift 2 support
• Objective-C module support

Improved data formatters
• Vector types in Objective-C and Swift
• Custom data formatters written in Swift

Address Sanitizer integration

memory history
memory read
memory write
memory find
Xcode 7

Improvements in expression evaluation
  • Swift 2 support
  • Objective-C module support

Improved data formatters
  • Vector types in Objective-C and Swift
  • Custom data formatters written in Swift

Address Sanitizer integration

Type lookup command
  • Built-in type documentation
Type Lookup
Type Lookup

(lldb)
Type Lookup

(lldb) type lookup ErrorType

protocol ErrorType {
    var _domain: Swift.String { get }
    var _code: Swift.Int { get }
}

(lldb)
Type Lookup

(lldb) type lookup ErrorType
protocol ErrorType {
    var _domain: Swift.String { get }
    var _code: Swift.Int { get }
}

(lldb) type lookup Comparable
protocol Comparable : _Comparable, Equatable {
    func <=(lhs: Self, rhs: Self) -> Swift.Bool
    func >=(lhs: Self, rhs: Self) -> Swift.Bool
    func >(lhs: Self, rhs: Self) -> Swift.Bool
}
Type Lookup

```swift
(protocol ErrorType {
    var _domain: Swift.String { get }
    var _code: Swift.Int { get }
})

(protocol Comparable : _Comparable, Equatable {
    func <=(lhs: Self, rhs: Self) -> Swift.Bool
    func >=(lhs: Self, rhs: Self) -> Swift.Bool
    func >(lhs: Self, rhs: Self) -> Swift.Bool
})
```
Compilers in LLDB
Making Swift and Objective-C better and easier to debug
Overview

Objective-C expression improvements
Swift error-handling support
Overview

Objective-C expression improvements
Swift error-handling support
Objective-C and Swift in LLDB
Objective-C and Swift in LLDB
Objective-C and Swift in LLDB
Objective-C and Swift in LLDB
How Swift Works in LLDB
How Swift Works in LLDB

(lldb) p for i in (0..<5) { print(i) }
How Swift Works in LLDB

(llldb) p for i in (0..<5) { print(i) }
How Swift Works in LLDB

```swift
(lldb) p for i in (0..<5) { print(i) }
0
1
2
3
4
```

Expression -- p
How Swift Works in LLDB

```swift
for i in (0..<5) { print(i) }
```

(lldb) `p for i in (0..<5) { print(i) }`

0
1
2
3
4
How Swift Works in LLDB

(lldb) p for i in (0..<5) { print(i) }

0
1
2
3
4
How Swift Works in LLDB

```
(llldb) p for i in (0..<5) { print(i) }
```

0
1
2
3
4
How Swift Works with Your Variables

Swift

LLDB

Added Code

Your Program

(lldb)
How Swift Works with Your Variables

 próp for i in (0..<3) { print(names[i]) }
How Swift Works with Your Variables

```swift
(lldb) p for i in (0..<3) { print(names[i]) }
Kate
Sean
Enrico
```
How Swift Works with Your Variables

(lldb) p for i in (0..<3) { print(names[i]) }

Kate
Sean
Enrico
How Swift Works with the SDK

Swift

LLDB

lldb

Added Code

Data

Your Program
How Swift Works with the SDK

(lldb) p NSApplication.sharedApplication()
How Swift Works with the SDK

(lldb) p NSApplication.sharedApplication()

(NSApplication) $R1 = 0x000060000100240 {
    AppKit.NSResponder = {
        NSObject = {
            isa = NSApplication
        }
        _nextResponder = nil
    }
}

(lldb) p NSApplication.sharedApplication()
How Swift Works with the SDK

```lldb
p NSApplication.sharedApplication()
```

```swift
(NSApplication) $R1 = 0x000060000100240 {
    AppKit.NSResponder = {
        NSObject = {
            isa = NSApplication
        }
        _nextResponder = nil
    }
}
```
How Swift Works with the SDK

(lldb) `p NSApplication.sharedApplication()`

(NSApplication) $R1 = 0x0000600000100240 {
    AppKit.NSResponder = {
        NSObject = {
            isa = NSApplication
        }
        _nextResponder = nil
    }
}
It’s All Automatic!
It’s All Automatic!

...in Swift
Let’s Try the Same Thing in Objective-C!

(lldb)
Let’s Try the Same Thing in Objective-C!

(llldb) p NSLog(@“%d”, i)
Let's Try the Same Thing in Objective-C!

(lldb) p NSLog(@"%d", i)
error: 'NSLog' has unknown return type;
   cast the call to its declared return type
error: 1 errors parsing expression
Let’s Try the Same Thing in Objective-C!

(lldb) p NSLog(@“%d”, i)

error: 'NSLog' has unknown return type;
    cast the call to its declared return type
error: 1 errors parsing expression

How it’s defined

void NSLog(NSString *format, ...)  

What LLDB sees

??? NSLog(...)
Let’s Try the Same Thing in Objective-C!

(lldb) p NSLog(@“%d”, i)
Let’s Try the Same Thing in Objective-C!

(llldb)
Let's Try the Same Thing in Objective-C!

(lldb) p NSMakeRect(0, 0, 10, 10)
Let’s Try the Same Thing in Objective-C!

(lldb) p NSMakeRect(0, 0, 10, 10)

error: use of undeclared identifier ‘NSMakeRect’
error: 1 errors parsing expression
Let’s Try the Same Thing in Objective-C!

```
(lldb) p NSMakeRect(0, 0, 10, 10)
error: use of undeclared identifier ‘NSMakeRect’
error: 1 errors parsing expression
```

<table>
<thead>
<tr>
<th>How it’s defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS_INLINE NSRect NSMakeRect( CGFloat, CGFloat, CGFloat, CGFloat, CGFloat)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What LLDB sees</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS_INLINE NSRect NSMakeRect( CGFloat, CGFloat, CGFloat, CGFloat, CGFloat)</td>
</tr>
</tbody>
</table>
Let’s Try the Same Thing in Objective-C!

(llldb) p NSMakeRect(0, 0, 10, 10)
(NSRect) $0 = (origin = (x = 0, y = 0), size = (width = 10, height = 10))
Let’s Try the Same Thing in Objective-C!

(lldb)
Let’s Try the Same Thing in Objective-C!

(lldb) p [NSApplication sharedApplication]..undoManager
Let’s Try the Same Thing in Objective-C!

(lldb) p [NSApplication sharedApplication].undoManager
error: property ‘undoManager’ not found on object of type ‘id’
error: 1 errors parsing expression
Let’s Try the Same Thing in Objective-C!

(lldb) p [NSApplication sharedApplication].undoManager
error: property ‘undoManager’ not found on object of type ‘id’
error: 1 errors parsing expression

<table>
<thead>
<tr>
<th>How it’s defined</th>
<th>-(NSApplication*)sharedApplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>What LLDB sees</td>
<td>-(id)sharedApplication</td>
</tr>
</tbody>
</table>
Let’s Try the Same Thing in Objective-C!

(lldb) p [NSApplication sharedApplication].undoManager
(NSUndoManager * __nullable) $1 = nil
“The Information Is Right There!”
“The Information Is Right There!”

Your Code

names, MyFunc(), MyView
"The Information Is Right There!"

Your Code

names, MyFunc(), MyView

SDK Functions

NSLog, NSMakeRect
“The Information Is Right There!”

<table>
<thead>
<tr>
<th>Your Code</th>
<th>names, MyFunc(), MyView</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK Functions</td>
<td>NSLog, NSMakeRect</td>
<td>✗</td>
</tr>
<tr>
<td>SDK Classes</td>
<td>NSView, NSApplication</td>
<td>✓  *</td>
</tr>
</tbody>
</table>
"The Information Is Right There!"

<table>
<thead>
<tr>
<th>Category</th>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your Code</td>
<td>names, MyFunc(), MyView</td>
<td>✅</td>
</tr>
<tr>
<td>SDK Functions</td>
<td>NSLog, NSMakeRect</td>
<td>✗</td>
</tr>
<tr>
<td>SDK Classes</td>
<td>NSView, NSApplication</td>
<td>✅</td>
</tr>
<tr>
<td>SDK Constants</td>
<td>NSASCIIStringEncoding</td>
<td>✗</td>
</tr>
</tbody>
</table>
"The Information Is Right There!"

<table>
<thead>
<tr>
<th>Category</th>
<th>Example</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your Code</td>
<td>names, MyFunc(), MyView</td>
<td>✔️</td>
</tr>
<tr>
<td>SDK Functions</td>
<td>NSLog, NSMakeRect</td>
<td>✗</td>
</tr>
<tr>
<td>SDK Classes</td>
<td>NSObject, NSWindow</td>
<td>✔️</td>
</tr>
<tr>
<td>SDK Constants</td>
<td>NSASCIIStringEncoding</td>
<td>✗</td>
</tr>
<tr>
<td>Macros</td>
<td>INT_MAX, MAX()</td>
<td>✗</td>
</tr>
</tbody>
</table>
"The Information Is Right There!"

<table>
<thead>
<tr>
<th>Your Code</th>
<th>names, MyFunc(), MyView</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK Functions</td>
<td>NSLog, NSMakeRect</td>
</tr>
<tr>
<td>SDK Classes</td>
<td>NSView, NSApplication</td>
</tr>
<tr>
<td>SDK Constants</td>
<td>NSASCIIStringEncoding</td>
</tr>
<tr>
<td>Macros</td>
<td>INT_MAX, MAX()</td>
</tr>
</tbody>
</table>
"The Information Is Right There!"

<table>
<thead>
<tr>
<th>Your Code</th>
<th>names, MyFunc(), MyView</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK Functions</td>
<td>NSLog, NSMakeRect</td>
<td>✓</td>
</tr>
<tr>
<td>SDK Classes</td>
<td>NSObject, NSApplication</td>
<td>✓</td>
</tr>
<tr>
<td>SDK Constants</td>
<td>NSASCIIStringEncoding</td>
<td>✓</td>
</tr>
<tr>
<td>Macros</td>
<td>INT_MAX, MAX()</td>
<td>✓</td>
</tr>
</tbody>
</table>
expr @import UIKit
Overview

Objective-C expression improvements

Swift error-handling support
Handling Errors in Swift Expressions
Handling Errors in Swift Expressions

Expressions can handle Swift errors
Handling Errors in Swift Expressions

Expressions can handle Swift errors

• No need to “try” in LLDB
Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to “try” in LLDB
Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to “try” in LLDB

(llldb) `expr ThisFunctionThrows()`
Expressions can handle Swift errors

- No need to “try” in LLDB

```lldb
(expr ThisFunctionThrows())
(a.EnumError) $E0 = SeriousError
```
Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to “try” in LLDB

```lldb
(expr ThisFunctionThrows() (a.EnumError) $E0 = SeriousError

- Works in the REPL, too!
Handling Errors in Swift Expressions

Expressions can handle Swift errors

• No need to “try” in LLDB

(lldb) expr ThisFunctionThrows()
(a.EnumError) $E0 = SeriousError

• Works in the REPL, too!

1>
Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to “try” in LLDB

```lldb
(expr ThisFunctionThrows()
(a.EnumError) $E0 = SeriousError
```

- Works in the REPL, too!

```1> ThisFunctionThrows()```
Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to “try” in LLDB

  (lldb) expr ThisFunctionThrows()

  (a.EnumError) $E0 = SeriousError

- Works in the REPL, too!

  1> ThisFunctionThrows()

  $E0: EnumError = MyError
Stopping When Swift Errors Occur
Stopping When Swift Errors Occur

Use breakpoints
Stopping When Swift Errors Occur

Use breakpoints

• On Objective-C exceptions
Stopping When Swift Errors Occur

Use breakpoints
  • On Objective-C exceptions

(lldb)
Stopping When Swift Errors Occur

Use breakpoints
• On Objective-C exceptions

(llldb) br s -E objc
Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

(lldb) br s -E objc
Stopping When Swift Errors Occur

Use breakpoints

• On Objective-C exceptions

(lldb) br s -E objc

Breakpoint 1: where = libobjc.A.dylib`objc_exception_throw,
    address = 0x00007fff9046bd18
Stopping When Swift Errors Occur

Use breakpoints

• On Objective-C exceptions

(lldb) br s -E objc

Breakpoint 1: where = libobjc.A.dylib`objc_exception_throw,
   address = 0x00007fff9046bd18

• On Swift errors
Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions
  
  (lldb) br s -E objc

  Breakpoint 1: where = libobjc.A.dylib`objc_exception_throw,
  address = 0x00007fff9046bd18

- On Swift errors
  
  (lldb)
Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions
  ```lldb
  br s -E objc
  Breakpoint 1: where = libobjc.A.dylib\objc_exception_throw,
  address = 0x00007fff9046bd18
  ```

- On Swift errors
  ```lldb
  br s -E swift
  ```
Stopping When Swift Errors Occur

Use breakpoints

• On Objective-C exceptions
  `(lldb) br s -E objc
  Breakpoint 1: where = libobjc.A.dylib`objc_exception_throw,
  address = 0x00007fff9046bd18

• On Swift errors
  `(lldb) br s -E swift
  Breakpoint 1: where = libswiftCore.dylib`swift_willThrow,
  address = 0x00000001001f71e0

`breakpoint set -E language`
Stopping on Specific Errors
Stopping on Specific Errors

Supported for Swift
Stopping on Specific Errors

Supported for Swift

(lldb)
Stopping on Specific Errors

Supported for Swift

```
(lldb) br s -E swift -O EnumError
Breakpoint 1: where = libswiftCore.dylib\swift\willThrow,
    address = 0x00000001001f71e0
(lldb)
```
Stopping on Specific Errors

Supported for Swift

(lldb) `breakpoint set -E swift -O EnumError`

Breakpoint 1: where = libswiftCore.dylib`swift_willThrow,
address = 0x00000001001f71e0

(lldb) `breakpoint set -E language -O type-name`
Stopping on Specific Errors

Supported for Swift

```
(lldb) br s -E swift -O EnumError
Breakpoint 1: where = libswiftCore.dylib`swift_willThrow,
                  address = 0x00000001001f71e0
(lldb) continue
  25       if input > 100
  26       {
-> 27           throw EnumError.ImportantError
  28       }
  29       else if input > 10
  30       {
```
Of Course, You Can Still Catch Them
Of Course, You Can Still Catch Them

1> import Foundation
2>
Of Course, You Can Still Catch Them

1> import Foundation
2> do {
    throw NSError(domain: "My domain", code: 0,
                 userInfo:nil)
} catch let x {
    print(x)
}
Of Course, You Can Still Catch Them

```swift
import Foundation

do {
    throw NSError(domain: "My domain", code: 0,
                   userInfo:nil)
} catch let x {
    print(x)
}

Error Domain=My domain Code=0
"The operation couldn’t be completed. (My domain error 0.)"
```
Presentation Is Everything

Formatted data is comprehended data
Examining Data
Examining Data

frame variable
Examining Data

frame variable

expression --
Examining Data

frame variable

expression -- p
Examining Data

frame variable

expression -- p

expression -0 --
Examining Data

frame variable

expression -- p

expression -O -- po
The “Frame Variable” Command
The “Frame Variable” Command

frame variable
The “Frame Variable” Command

(frame) frame variable
(Int, String) myTuple = (0 = 12, 1 = “Hello World”)
(Int) theYear = 1984
The “Frame Variable” Command

(lldb) frame variable
(Int, String) myTuple = (0 = 12, 1 = ”Hello World”)  
(Int) theYear = 1984

(lldb) frame variable theYear
(Int) theYear = 1984
The “Frame Variable” Command

(lldb) frame variable
(Int, String) myTuple = (0 = 12, 1 = “Hello World”)  
(Int) theYear = 1984

(lldb) frame variable theYear
(Int) theYear = 1984

(lldb) frame variable —--format hex theYear
(Int) theYear = 0x00000000000007c0
The “Frame Variable” Command

(frame) frame variable
(Int, String) myTuple = (0 = 12, 1 = “Hello World”)
(Int) theYear = 1984

(frame) frame variable theYear
(Int) theYear = 1984

(frame) frame variable —-format hex theYear
(Int) theYear = 0x00000000000007c0
The “Expression” Command
The “Expression” Command
The “Expression” Command

(lldb) p theYear + 1
(Int) $R0 = 1985
The “Expression” Command

(lldb) p theYear + 1
(Int) $R0 = 1985

(lldb) p String($R0 + 1)
(String) $R1 = “1986”
The “Expression” Command

(lldb) p theYear + 1
(Int) $R0 = 1985

(lldb) p String($R0 + 1)
(String) $R1 = “1986”

(lldb) expr --format hex -- [1,2,3]
([Int]) $R0 = 3 values {
  [0] = 0x0000000000000001
  [1] = 0x0000000000000002
  [2] = 0x0000000000000003
}
The “Expression” Command

(lldb) p theYear + 1
(Int) $R0 = 1985

(lldb) p String($R0 + 1)
(String) $R1 = “1986”

(lldb) expr --format hex -- [1,2,3]
([Int]) $R0 = 3 values {
    [0] = 0x000000000000000000000001
    [1] = 0x000000000000000000000002
    [2] = 0x000000000000000000000003
}

children
The “po” Command
The “po” Command

expression -O -- po
The “po” Command

(lldb) po [[MyView alloc] initWithFrame: myFrame]
<MyView: 0x101405110>
The “po” Command

(lldb) po [[MyView alloc] initWithFrame: myFrame]
<MyView: 0x101405110>

(lldb) po @[1, 2, 3]
<__NSArrayI 0x10050b2a0>(
1,
2,
3
)
The “po” Command

(lldb) po [[MyView alloc] initWithFrame: myFrame]
<MyView: 0x101405110>

(lldb) po @[1, 2, 3]
<__NSArrayI 0x10050b2a0>(
  1,
  2,
  3
)

(lldb) po @“Hello World”
Hello World
Formatting Models
## Formatting Models

<table>
<thead>
<tr>
<th>Frame Variable</th>
<th>Expression -- p</th>
<th>Expression -O -- po</th>
</tr>
</thead>
<tbody>
<tr>
<td>does not run code</td>
<td>runs your code</td>
<td>runs your code</td>
</tr>
<tr>
<td>uses LLDB formatters</td>
<td>uses LLDB formatters</td>
<td>+ code to format object</td>
</tr>
</tbody>
</table>
## Formatting Models

<table>
<thead>
<tr>
<th>frame variable</th>
<th>expression --</th>
<th>p</th>
<th>expression -O --</th>
<th>po</th>
</tr>
</thead>
<tbody>
<tr>
<td>does not run code</td>
<td>runs your code</td>
<td>runs your code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uses LLDB formatters</td>
<td>uses LLDB formatters</td>
<td>+ code to format object</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Formatting Models

<table>
<thead>
<tr>
<th>Frame Variable</th>
<th>Expression --</th>
<th>P</th>
<th>Expression -O --</th>
<th>PO</th>
</tr>
</thead>
<tbody>
<tr>
<td>does not run code</td>
<td>runs your code</td>
<td>runs your code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uses LLDB formatters</td>
<td>uses LLDB formatters</td>
<td>+ code to format object</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Formatting Models
Out-of-process formatting

• Data and formatter are separate
• Easy access to debugger’s object model
• Easier to keep program state intact
Formatting Models

Out-of-process formatting
- Data and formatter are separate
- Easy access to debugger’s object model
- Easier to keep program state intact

In-process formatting
- Data and formatter live together
- Easy access to your object model
- Care required to keep program state intact
In-Process Swift Formatting
In-Process Swift Formatting

Powers playgrounds since day one
In-Process Swift Formatting

Powers playgrounds since day one

Now

• Public API
• Also powers LLDB “po” command
In-Process Swift Formatting
In-Process Swift Formatting

Four Swift protocols
In-Process Swift Formatting

Four Swift protocols

• CustomStringConvertible
• CustomDebugStringConvertible
• CustomPlaygroundQuickLookable
• CustomReflectable
In-Process Swift Formatting

Four Swift protocols
• CustomStringConvertible
• CustomDebugStringConvertible
• CustomPlaygroundQuickLookable
• CustomReflectable
Partial opt-in possible
CustomStringConvertible

String representation of the object
Used by the print() function, and string interpolation
CustomStringConvertible

String representation of the object
Used by the print() function, and string interpolation

```swift
struct BottlesOfBeer : CustomStringConvertible {
    var Count: Int
    var description: String {
        return “\(Count) bottles of beer on the wall”
    }
}
```
CustomDebugStringConvertible

Debugger-specific String representation of the object

Used by debugPrint() (and print() if necessary)
CustomDebugStringConvertible

Debugger-specific String representation of the object
Used by debugPrint() (and print() if necessary)

```swift
extension BottlesOfBeer : CustomDebugStringConvertible {
    var debugDescription: String {
        return "\(Count) bottles of stout on the wall."
    }
}
```
CustomPlaygroundQuickLookable

Graphical representation for display in playgrounds
import Cocoa

struct Person : CustomPlaygroundQuickLookable {

    func customPlaygroundQuickLook() -> PlaygroundQuickLook {
        let path = NSBezierPath(rect: NSRect(x: 0, y: 0, width: 30, height: 75))
        path.appendBezierPathWithRoundedRect(NSRect(x: 7.5, y: 75, width: 15, height: 30),
            xRadius: 10, yRadius: 10)
        path.moveToPoint(NSPoint(x: 30, y: 50))
        path.lineToPoint(NSPoint(x: 45, y: 40))
        path.moveToPoint(NSPoint(x: 0, y: 50))
        path.lineToPoint(NSPoint(x: 15, y: 40))
        path.moveToPoint(NSPoint(x: 5, y: 0))
        path.lineToPoint(NSPoint(x: 15, y: -10))
        path.lineToPoint(NSPoint(x: 25, y: 0))
        path.lineToPoint(NSPoint(x: 30, y: -10))
        return PlaygroundQuickLook(reflecting: path)
    }

    Person()
}

22 path elements
CustomReflectable

Allows vending an entirely custom children hierarchy
CustomReflectable

Allows vending an entirely custom children hierarchy

Returns a **Mirror** for an object

- Representation of an object’s structure
Temperature Data
Temperature Data

```swift
struct Timestamp {
    var Hour: Int
    var Minute: Int
}

struct TemperatureData {
    var Time: Timestamp
    var Temperature: Float
}
```
struct Timestamp {
    var Hour: Int
    var Minute: Int
}

struct TemperatureData {
    var Time: Timestamp
    var Temperature: Float
}

let Temps = [
    TemperatureData(Time: Timestamp(Hour: 6, Minute: 30), Temperature: 10),
    TemperatureData(Time: Timestamp(Hour: 18, Minute: 30), Temperature: 34)
]
Temperature Data
Temperature Data

(lldb) po Temps
Temperature Data

(lldb) po Temps

- 2 elements
  - [0] : TemperatureData
    - Time : Timestamp
      - Hour : 6
      - Minute : 30
      - Temperature : 10
  - [1] : TemperatureData
    - Time : Timestamp
      - Hour : 18
      - Minute : 30
      - Temperature : 34
Temperature Data

(lldb) po Temps
- 2 elements
  - [0] : TemperatureData
    - Time : Timestamp
      - Hour : 6
      - Minute : 30
      - Temperature : 10
  - [1] : TemperatureData
    - Time : Timestamp
      - Hour : 18
      - Minute : 30
      - Temperature : 34
Temperature Data

(lldb) po Temps

2 elements

[0] : TemperatureData
  Time : Timestamp
    - Hour : 6
    - Minute : 30
    - Temperature : 10

[1] : TemperatureData
  Time : Timestamp
    - Hour : 18
    - Minute : 30
    - Temperature : 34
Step 1: Time on One Line
extension Timestamp : CustomStringConvertible {
    var description: String {
        let formatter = NSDateFormatter()
        formatter.dateStyle = NSDateFormatterStyle.NoStyle
        formatter.timeStyle = NSDateFormatterStyle.ShortStyle
        formatter.timeZone = NSTimeZone(forSecondsFromGMT: 0)
        let date = NSDate(timeIntervalSince1970: NSTimeInterval(60 * (Minute + (60*Hour)))))
        return formatter.stringFromDate(date)
    }
}
Step 2: Fahrenheit Degrees
Step 2: Fahrenheit Degrees

```swift
extension TemperatureData : CustomReflectable {
    func customMirror() -> Mirror {
        return Mirror(self, children: [

            "Time" : "\(Time)",

            "Temp C" : Temperature,

            "Temp F" : 1.8*Temperature+32

        ])
    }
}
```
extension TemperatureData : CustomReflectable {
    func customMirror() -> Mirror {
        return Mirror(self, children: [

            "Time" : "\(Time)",

            "Temp C" : Temperature,

            "Temp F" : 1.8*Temperature+32

        ])
    }
}

CustomStringConvertible from previous slide
Temperature Data
Temperature Data

(lldb) po Temps
Temperature Data

(lldb) po Temps

2 elements

[0] : TemperatureData
- Time : “6:30 AM”
- Temp C : 10
- Temp F : 50

[1] : TemperatureData
- Time : “6:30 PM”
- Temp C : 34
- Temp F : 93.2
Temperature Data

(lldb) po Temps

 mắt 2 elements

мя[0] : TemperatureData
- Time : “6:30 AM”
- Temp C : 10
- Temp F : 50

мя[1] : TemperatureData
- Time : “6:30 PM”
- Temp C : 34
- Temp F : 93.2 -> That’s hot!
At Runtime Too!

Conformances can also be added while debugging
At Runtime Too!

Conformances can also be added while debugging but not changed
At Runtime Too!

Conformances can also be added while debugging but not changed.

They can also be added in the REPL.
And in Playgrounds!
And in Playgrounds!
Summary
Summary

More information available more often
Summary

More information available more often

• Objective-C runtime
Summary

More information available more often

- Objective-C runtime
- SDK modules
Summary

More information available more often

- Objective-C runtime
- SDK modules
- In-process formatting
More Information

LLDB Documentation
http://lldb.llvm.org

Swift Language Documentation
http://developer.apple.com/swift

Apple Developer Forums
http://developer.apple.com/forums

Stefan Lesser
Developer Tools Evangelist
slesser@apple.com
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authoring Rich Playgrounds</td>
<td>Presidio</td>
<td>Wednesday 10:00AM</td>
</tr>
<tr>
<td>Advanced Debugging and the Address Sanitizer</td>
<td>Mission</td>
<td>Friday 9:00AM</td>
</tr>
</tbody>
</table>